Java-Security Concepts

Ovidiu DOBRE

ovidiu.dobre@ifaedi.insa-lyon.fr

System Architecture Group

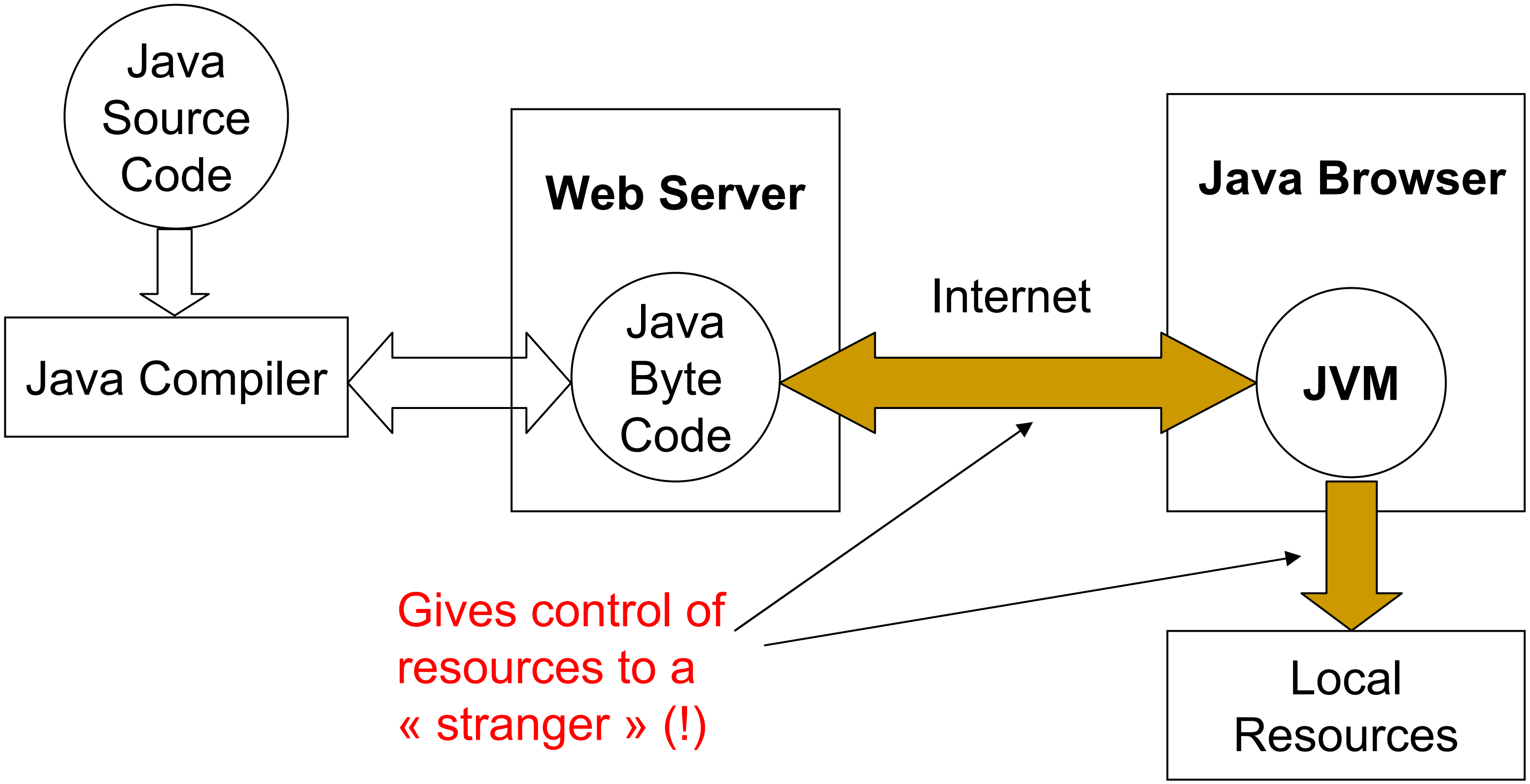
Department of Computer Science

University of Karlsruhe

Java-Security Concepts
Table of contents

I. Why Java needs security?

I. Why Java needs security?



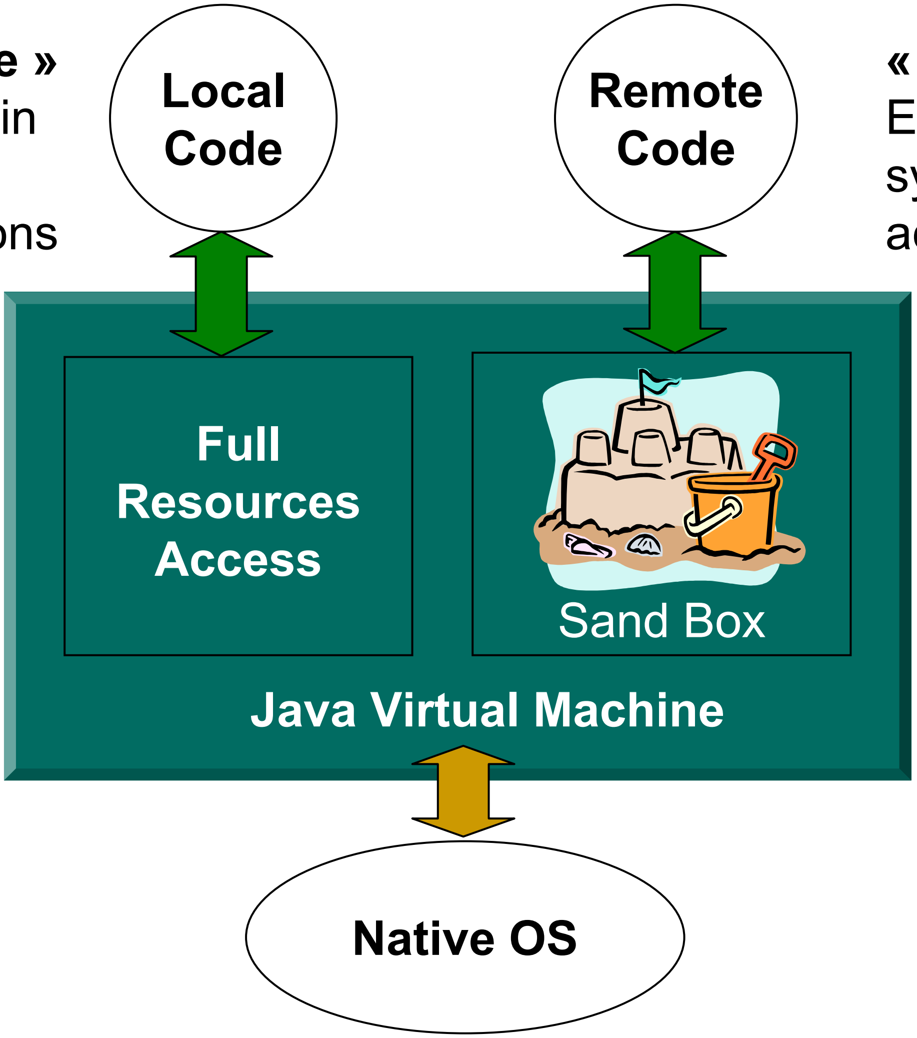
Java needs security !!!

II. How Java defines security?

Java 1.0 – The SandBox Model

« trusted code »
Includes build-in code and local Java applications

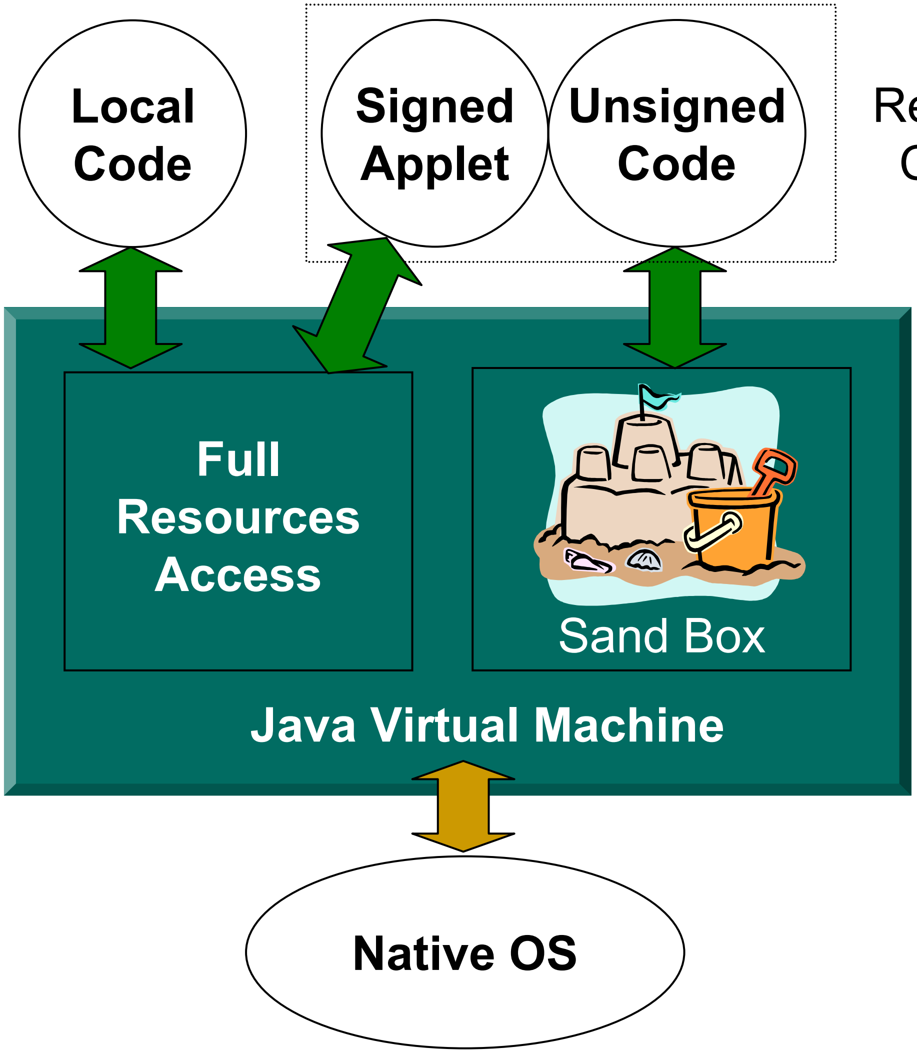
« untrusted code »
Extremely limited system resources access



Too restricting security model

Java 1.1 – The Signed Applet

« trusted code»
No changes from
Java 1.0

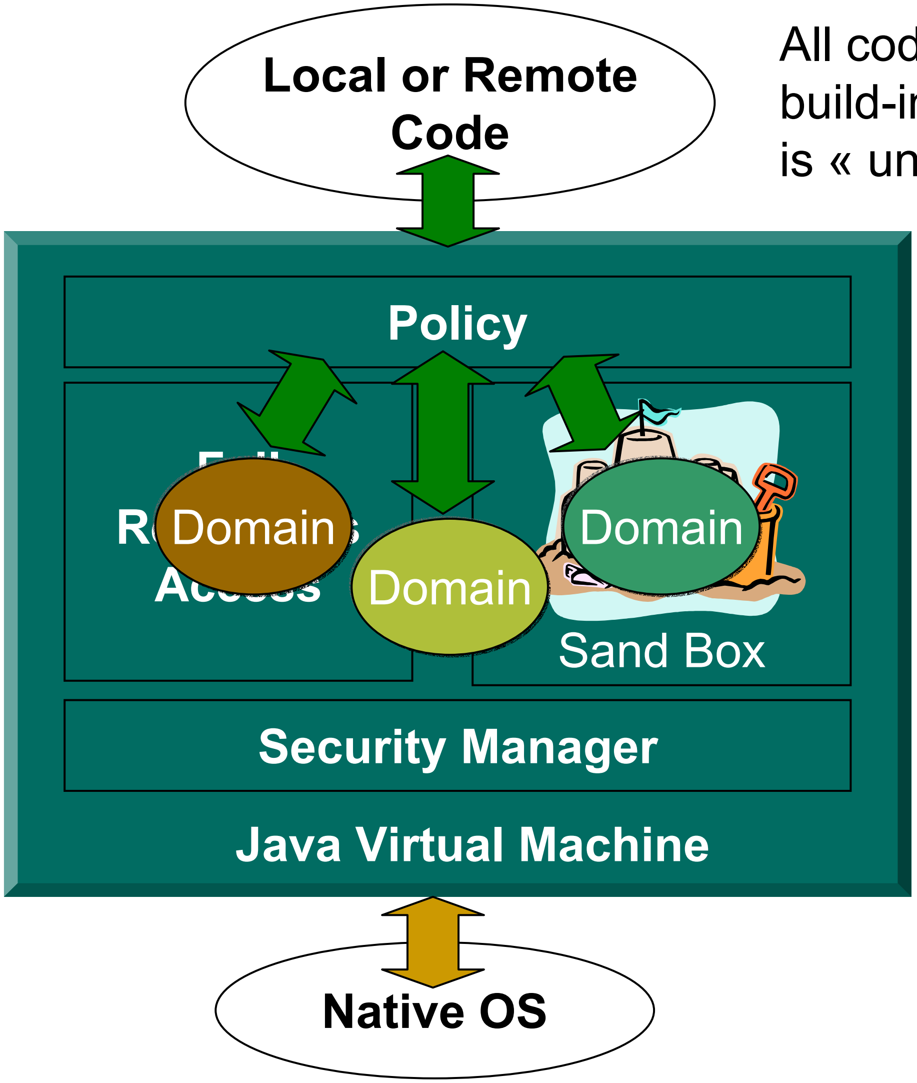


Remote
Code

Digitally signed applets become « trusted » => **FULL** functionalities for remote code

But granularity, « partially trusted » still not possible

Java 2 – Security Management



Local or Remote Code

Policy

Domain

Domain

Domain

Sand Box

Security Manager

Java Virtual Machine

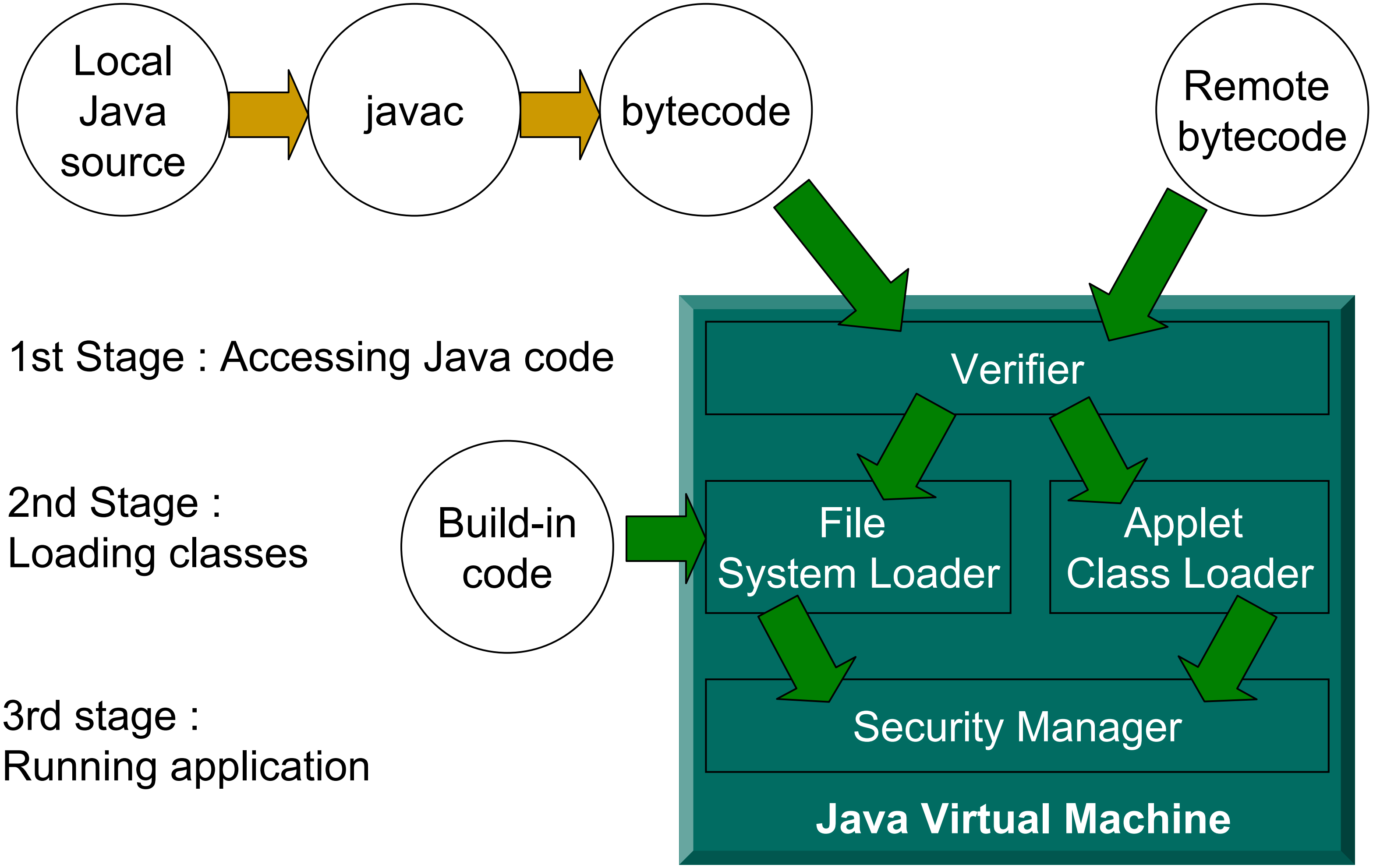
Native OS

All code except
build-in code
is « untrusted »

Fine-grained level
of access control

III. Java 2 Security Model

Java 2 Security Architecture

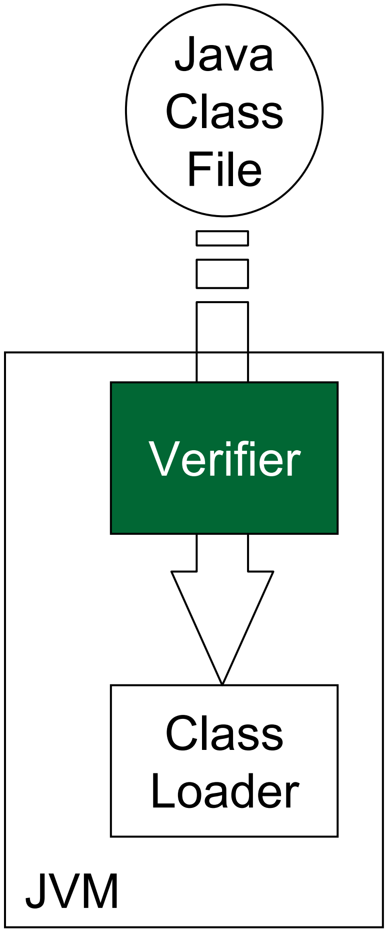


1st Stage : Accessing Java code

2nd Stage :
Loading classes

3rd stage :
Running application

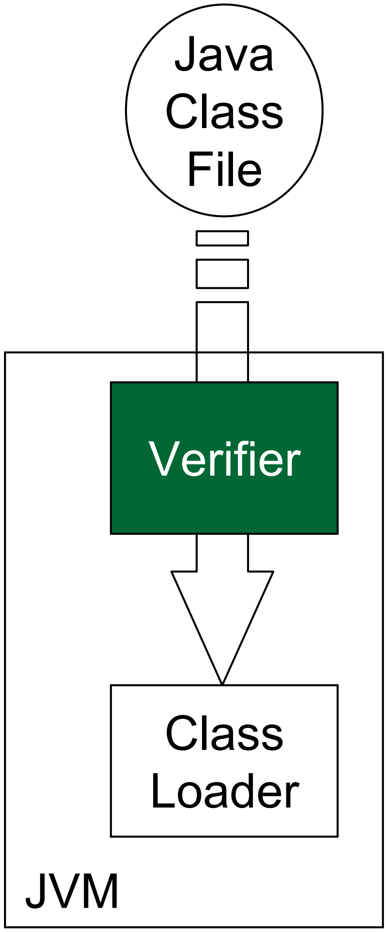
First Stage: Accessing Java Code
The Verifier (1)



Verifier Purpose

- « First gatekeeper in Java security model »
- All « untrusted » code is checked before loaded in the system

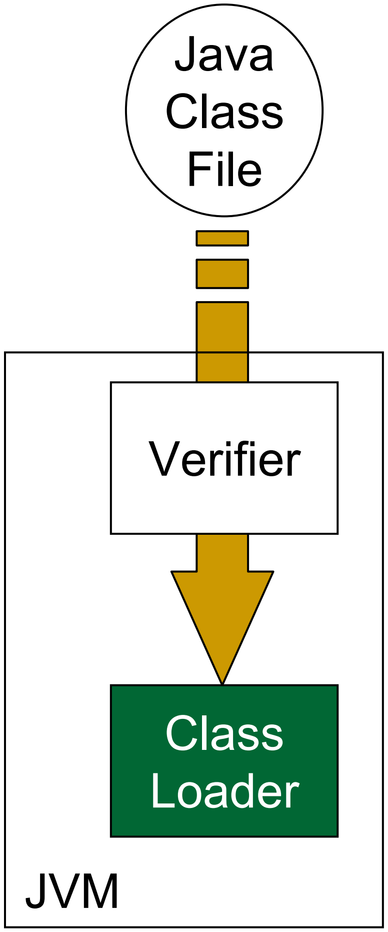
First Stage: Accessing Java Code
The Verifier (2)



Verifying process

- Step 1 : proper file format
- Step 2 : references (final methods and classes, super class, constants)
- **Step 3 : bytecode verification using data flow analysis**
  - Stack size
  - Register accesses
  - Method calls with correct arguments

Second Stage: Loading classes
The Class Loader (1)



Class Loader purpose

- « dynamic linking »

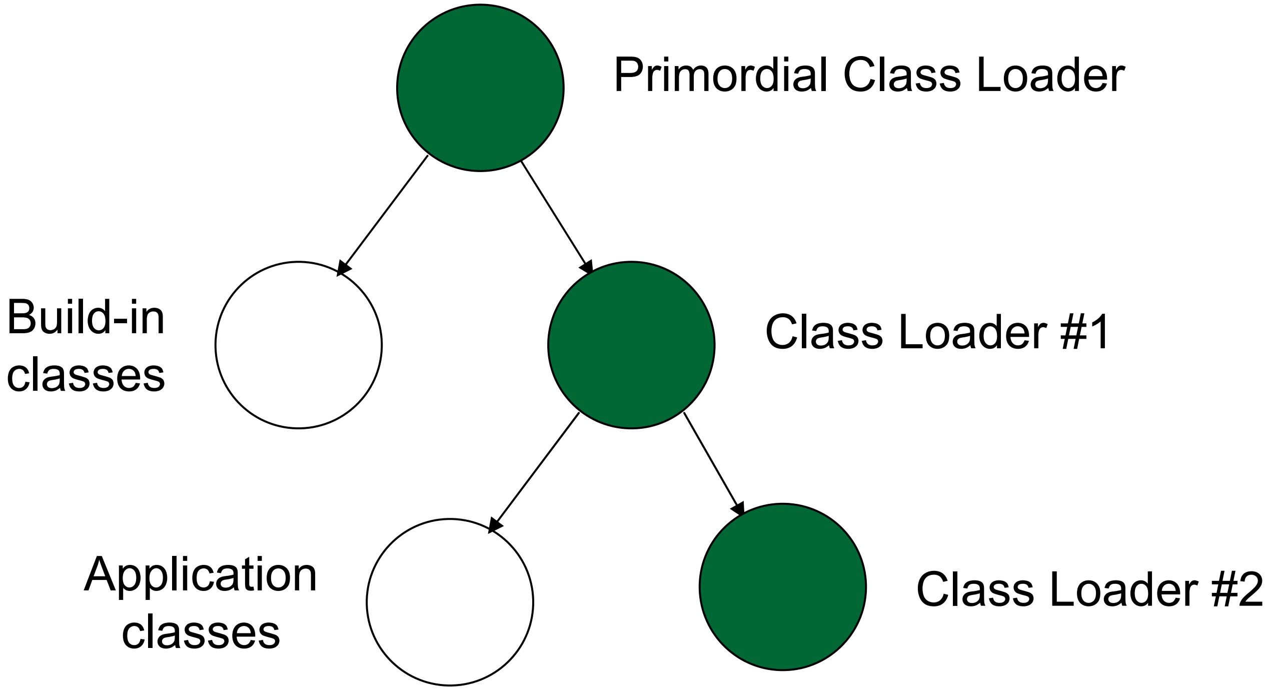
= loading classes at runtime (e.g. applets)

= defear loading classes until they are needed

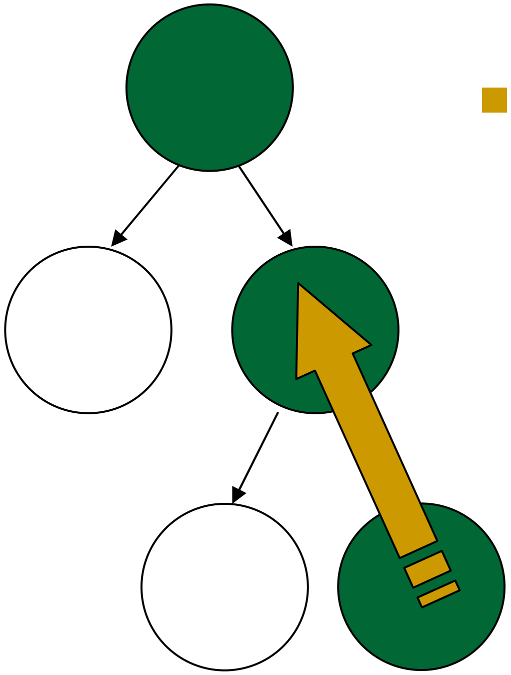
- Load from different locations
  - File System Loader
  - Applet Class Loader
- Prevent « class spoofing »
- Manage namespaces

Second Stage: Loading classes
The Class Loader (2)

Class Loader hierarchy



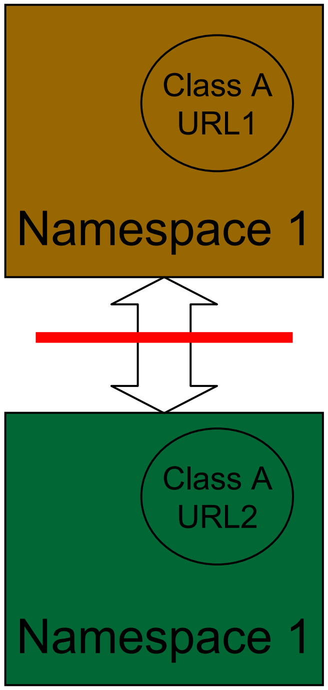
Second Stage: Loading classes
The Class Loader (3)



Preventing *class spoofing*

- « *class spoofing* » = pretend to be someone you're not in order to gain permissions
  - Avoid system class spoofing
  - Follow class loader hierarchy

Second Stage: Loading classes
The Class Loader (4)



Namespace management

- Let Java classes « see different views of the world »
- A class can « see » (or reference) only classes originating from the same location or the Java build-in classes

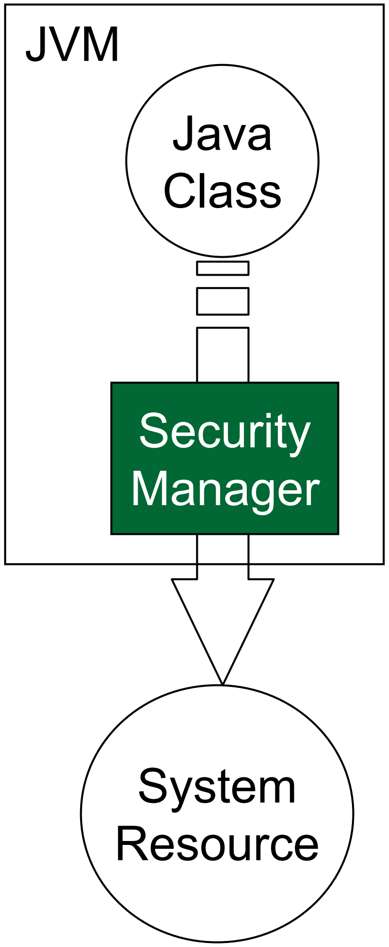
  => allows browsers to run applets with identical names as they originate from the different locations

Second Stage: Loading classes
The Class Loader (5)

Default Applet Security Rule

An applet cannot create a ClassLoader !

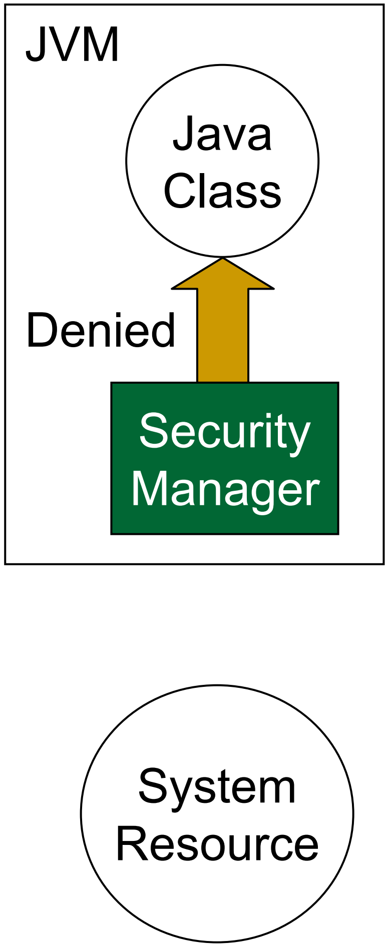
Third Stage: Running application
The Security Manager (1)



Security Manager Purpose

- Queried by JVM each time an « untrusted » code tries to access a system resource

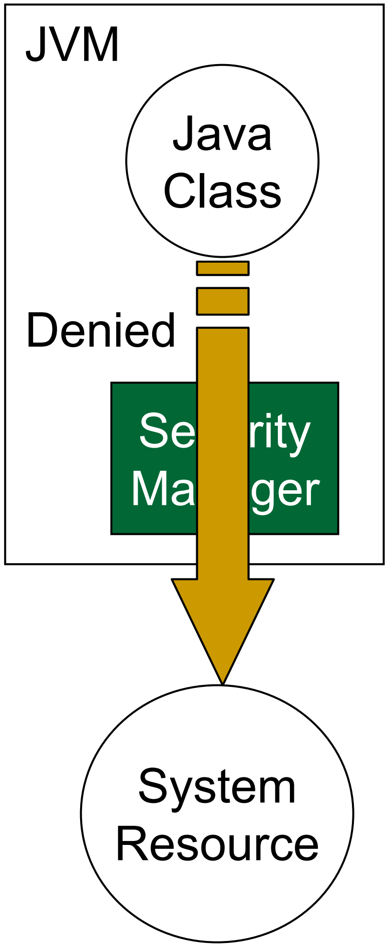
Third Stage: Running application
The Security Manager (1)



Security Manager Purpose

- Queried by JVM each time an « untrusted » code tries to access a system resource
- There are so-called *check methods* for every system resource (ex. *checkRead*, *checkWrite*) for user-level checks
    - If access is denied, a *SecurityException* is thrown

Third Stage: Running application
The Security Manager (1)



Security Manager Purpose

- Queried by JVM each time an « untrusted » code tries to access a system resource
- There are so-called *check methods* for every system resource (ex. *checkRead*, *checkWrite*) for user-level checks
  - If access is denied, a *SecurityException* is thrown
  - Otherwise the *check method* returns quietly

Third Stage: Running application
The Security Manager (2)

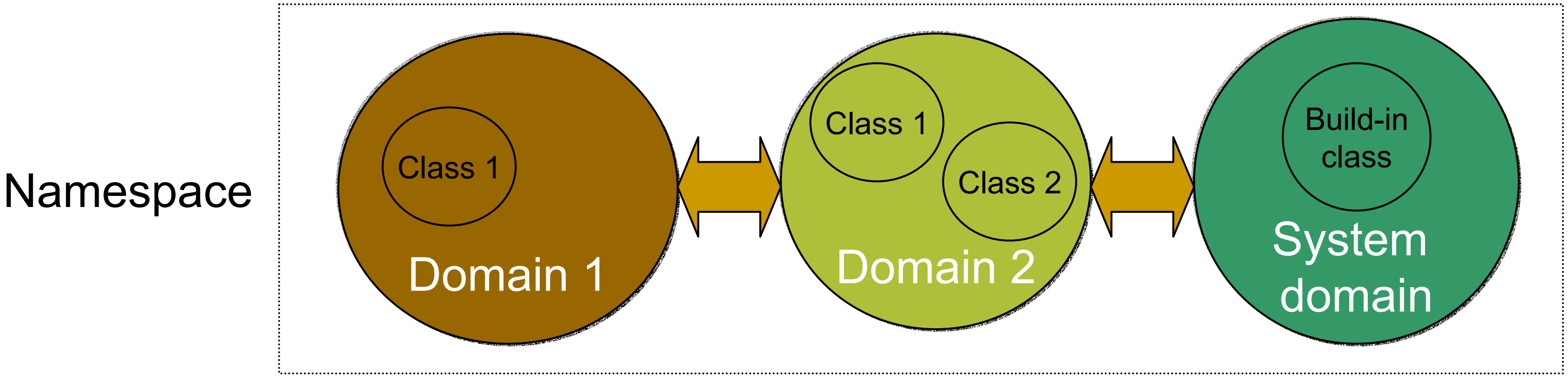
Policy

- Tells « which class has the right to do what » depending on the class identity
- Identity of a class = the URL + a set of certificates

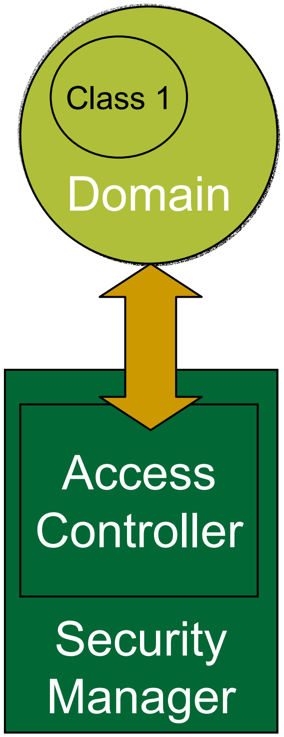
Third Stage: Running application
The Security Manager (3)

Protection Domains

- « protection domain » = a set of permissions established by the current *policy*
- Every class is assigned by the *policy* to « a protection domain »



Third Stage: Running application
The Security Manager (4)



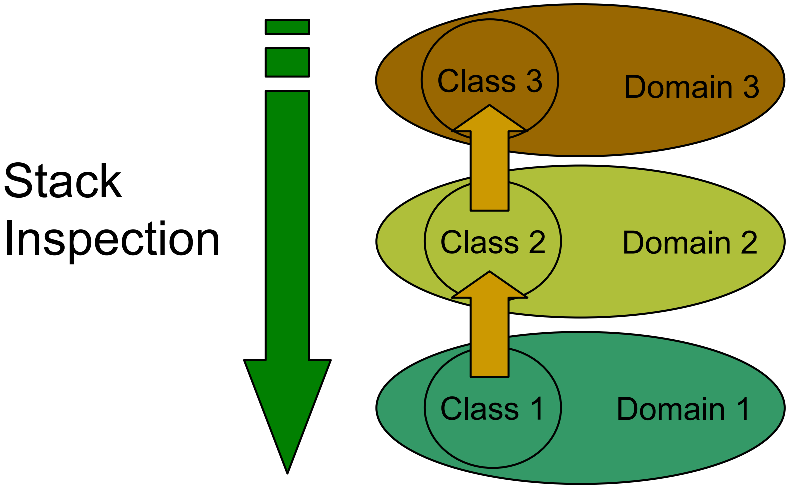
Decision making process

- *Security Manager* delegates all decisions to a component called *Access Controller*
- The *Access Controller* takes the decision according to :
  - the permissions of the *protection domain*
  - the « stack inspection »

Third Stage: Running application
The Security Manager (5)

Stack Inspection

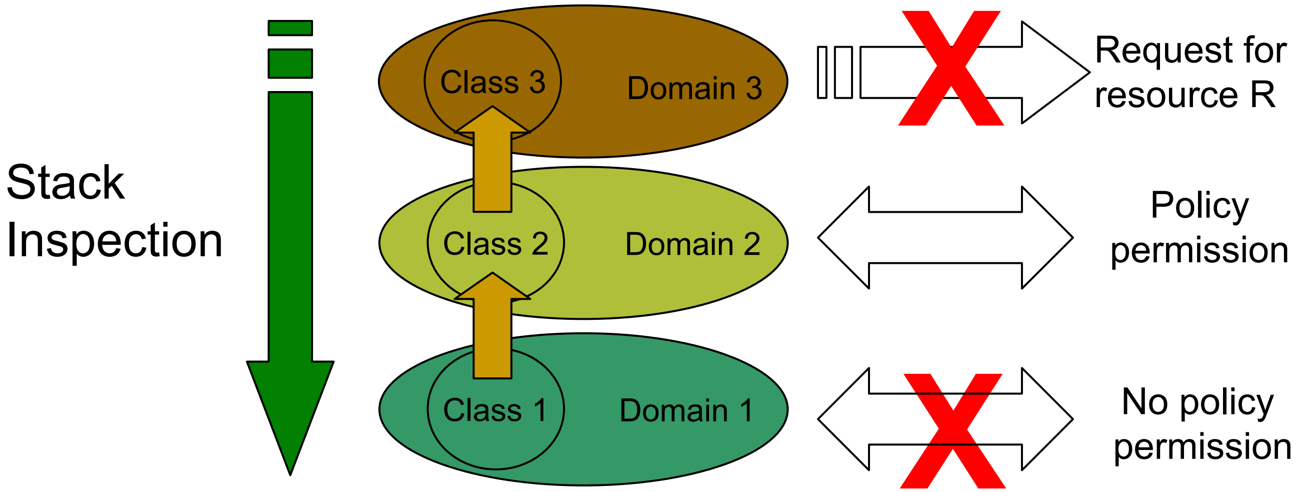
- An algorithm which checks the sequence of calling-classes for resource grant/denial



Third Stage: Running application
The Security Manager (6)

Stack Inspection : Deny Access

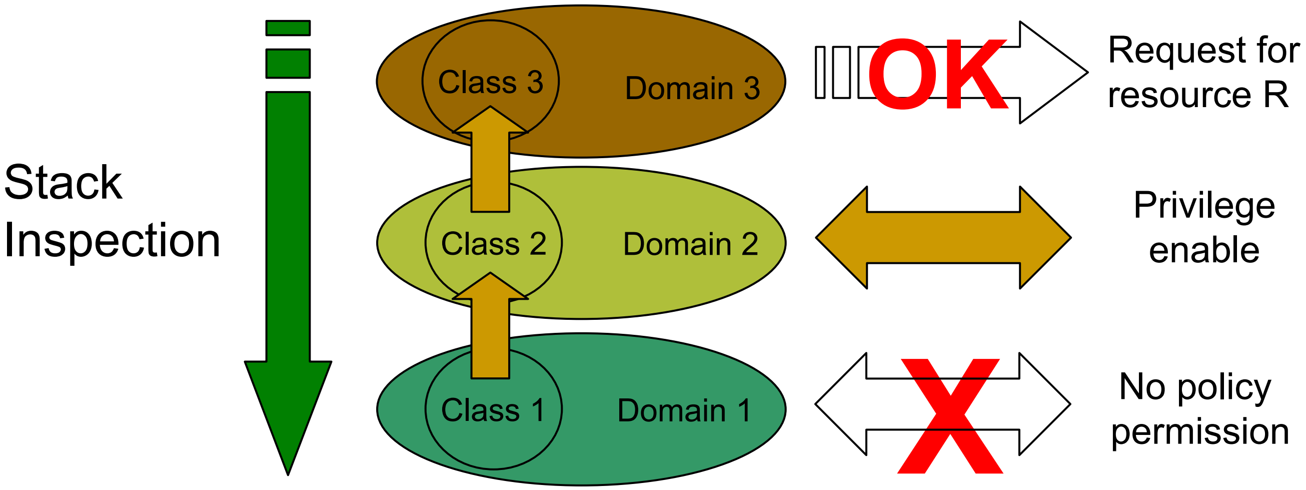
- Access is refused if a class with no policy permission is detected



Third Stage: Running application
The Security Manager (7)

Stack Inspection : Grant Access

- Access is granted if a class with resource-enabled privilege is detected



Third Stage: Running application
The Security Manager (8)

Default Applet Security Rule

An applet cannot create a Security Manager !

Customizing Java 2 Security Model

- All security components except the Verifier can be customized
  - e.g. Class Loader, Security Manager, Policy
- User-level customization : Policy (using Java Policy Tool)
- Developer-level customization : all security components except the Verifier

Is up to the user to set up a secure environment!

Java Security Classes

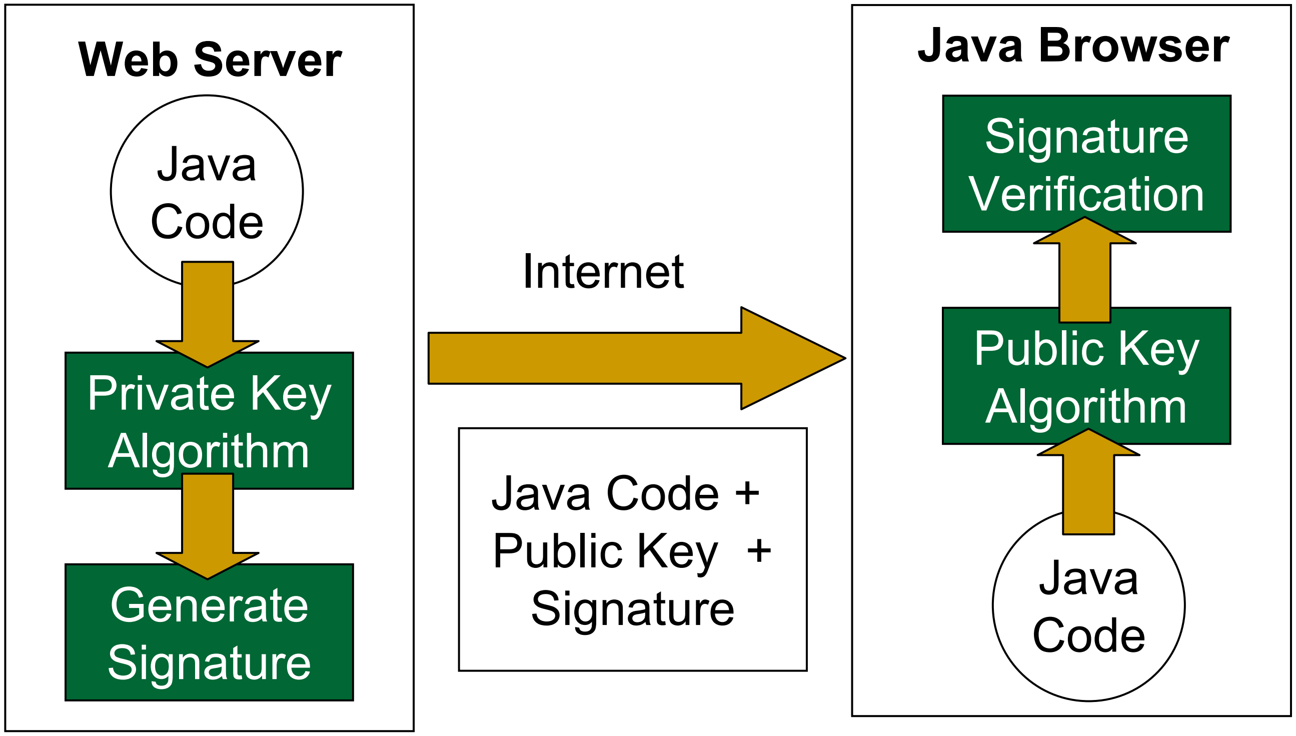
- For developer-level customization
- Included in *java.security* package
  - java.security.Permission
  - java.security.ProtectionDomain
  - java.security.Policy
  - java.security.SecureClassLoader
  - …
- Particular *permission* implementations
  - java.io.FilePermission
  - java.net.SocketPermission
  - …

Java 2 – Default Security

- By default, an applet cannot :
    - Access system files : create, read, write, delete, rename, check for existence of files or directories
    - Listen for or accept network connections on any port on the client system.
    - Create a top-level window without an untrusted window banner.
    - Obtain the user's username or home directory
    - Run any program on the client system
    - Make the Java interpreter exit
    - Create a Class Loader or a Security Manager

Cryptography, Signatures and Certificates (1)

- Ensures the identity of the sender and guarantees that the code is not modified during the transfer



Web Server

Java Code

Private Key Algorithm

Generate Signature

Internet

Java Code + Public Key + Signature

Java Browser

Signature Verification

Public Key Algorithm

Java Code

Cryptography, Signatures and Certificates (2)

- Signatures and public keys are usually delivered in a certificate (which contains certificate's issuer, serial number) delivered by a certificate authority

- Even certificates ensure the identity of the signer, they do NOT ensure that the code will well-behave!

- Therefore, the policy should be customized in order to allow/deny permissions to a particular signer.

IV. Famous Java Security Flaws

not for hacking purposes…

Case 1 - Jumping the Firewall (1)

Java Applet Security Rule:

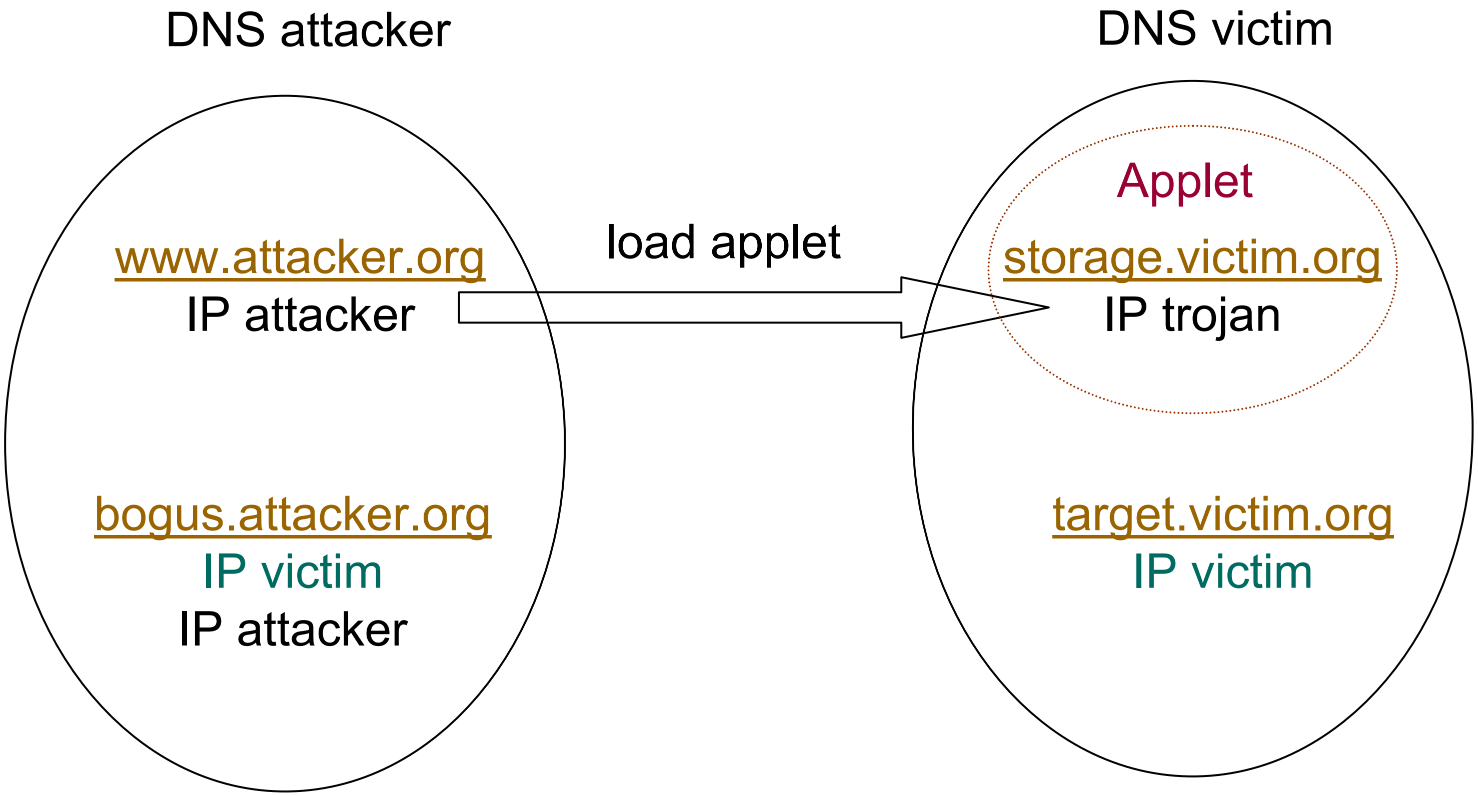
> « An applet may not open a network connection except back to the server from which it came »

This security rule was bypassed in 1996 by Steve Gibbons using a DNS security flaw

Hacking goal using this DNS security flaw :

Attack a machine inside a firewall using an applet running on other machine (the « Trojan » machine) inside the same firewall

Case 1 - Jumping the Firewall (2)



DNS attacker

www.attacker.org
IP attacker

bogus.attacker.org
IP victim
IP attacker

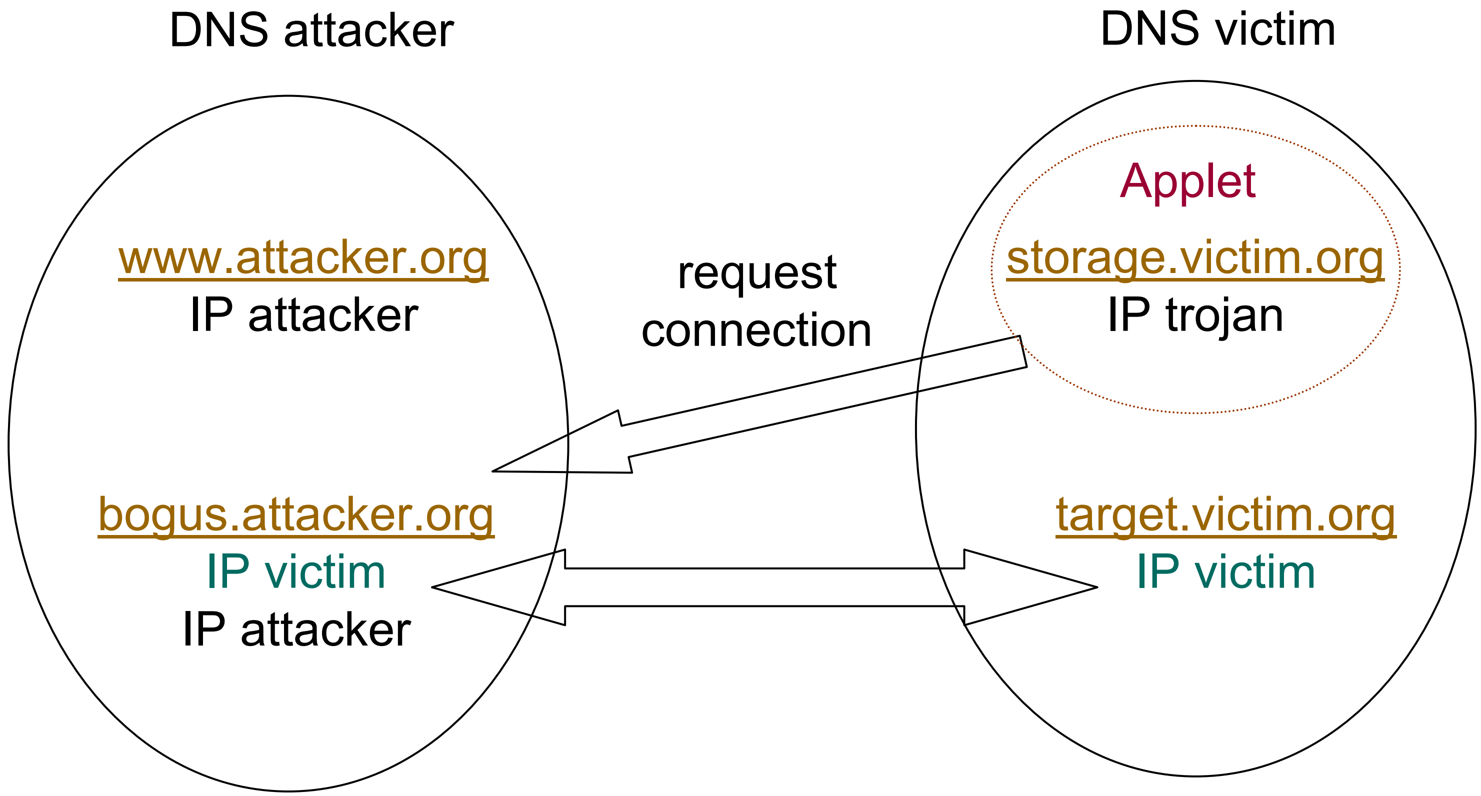
load applet

DNS victim

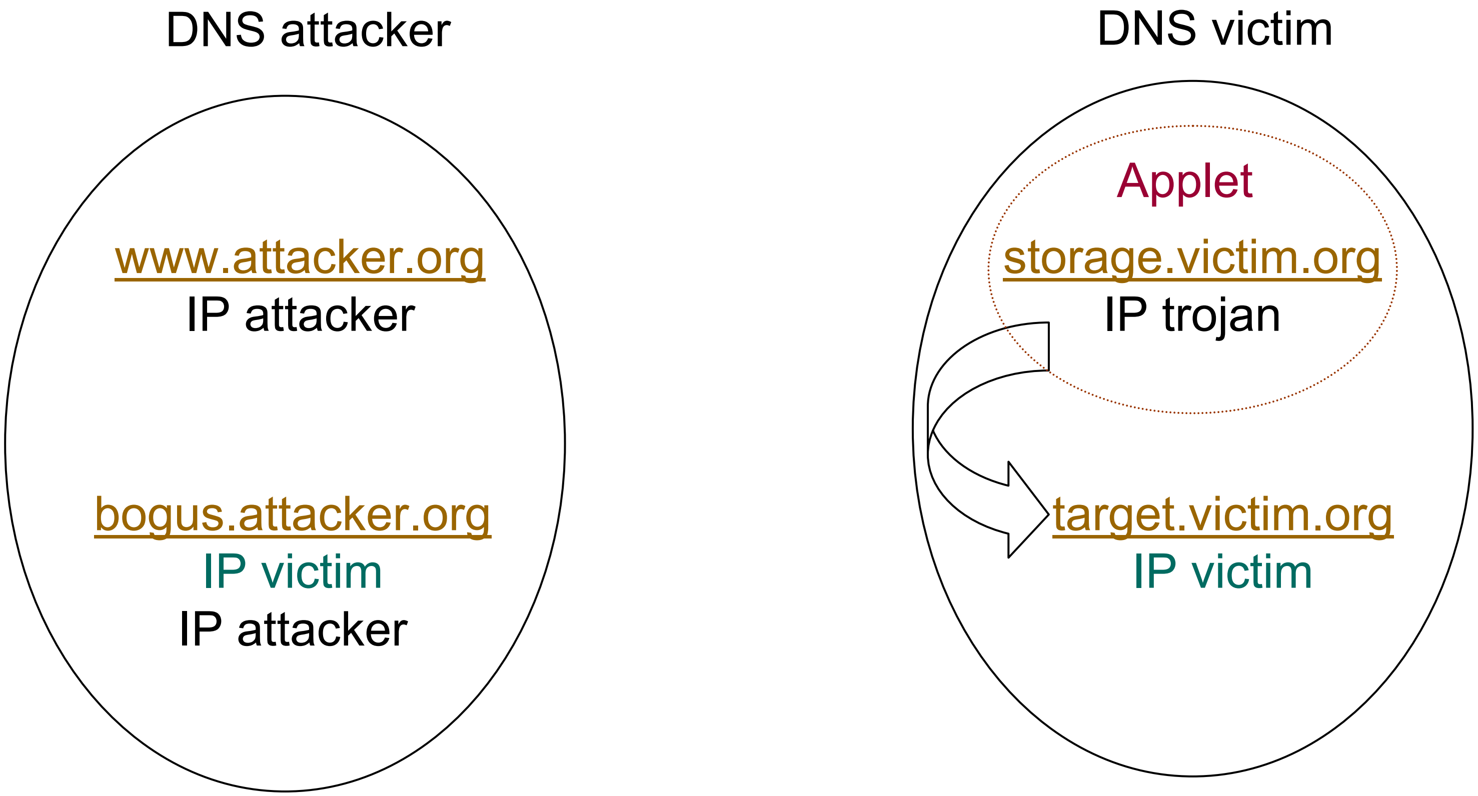
Applet
storage.victim.org
IP trojan

target.victim.org
IP victim

Case 1 - Jumping the Firewall (2)



Case 1 - Jumping the Firewall (2)



DNS attacker

www.attacker.org
IP attacker

bogus.attacker.org
IP victim
IP attacker

DNS victim

Applet

storage.victim.org
IP trojan

target.victim.org
IP victim

Fix ? Store IP address of Web server, NOT DNS name

Case 2 – Slash and Burn (1)

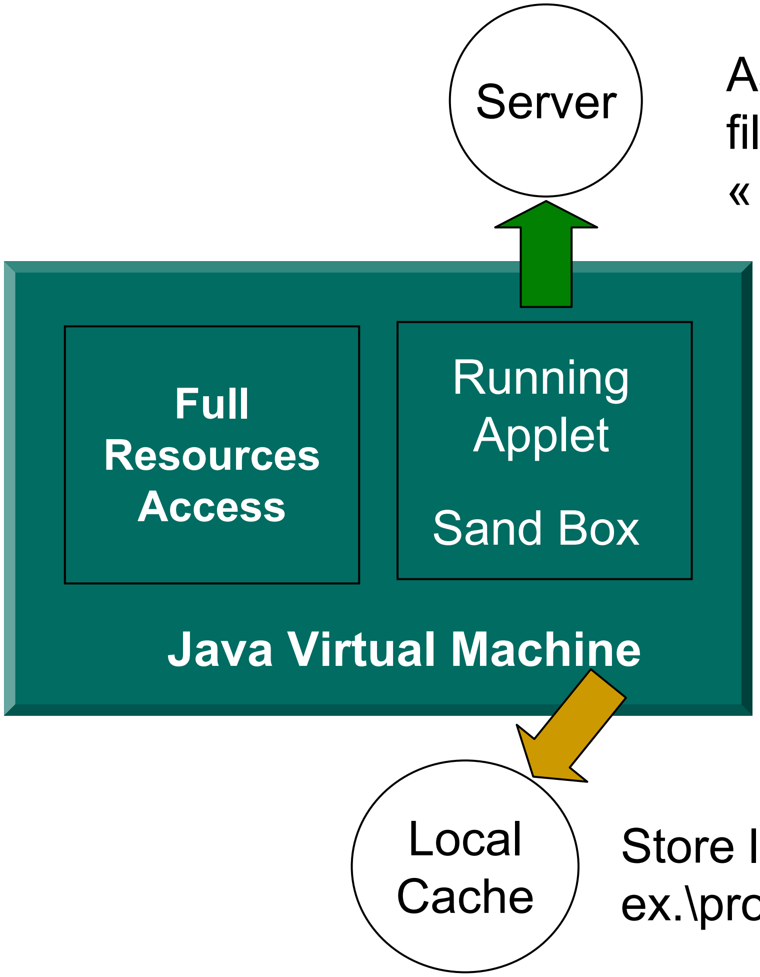
Dots & Slashes Rule:

« A class file is searched on local system using a dot (.) to slash (\ or /) conversion of class name »

ex. java.security.policy is searched as java\security\policy

This rule allowed to run an « intrusion » applet stored in cache, using FULL resources access

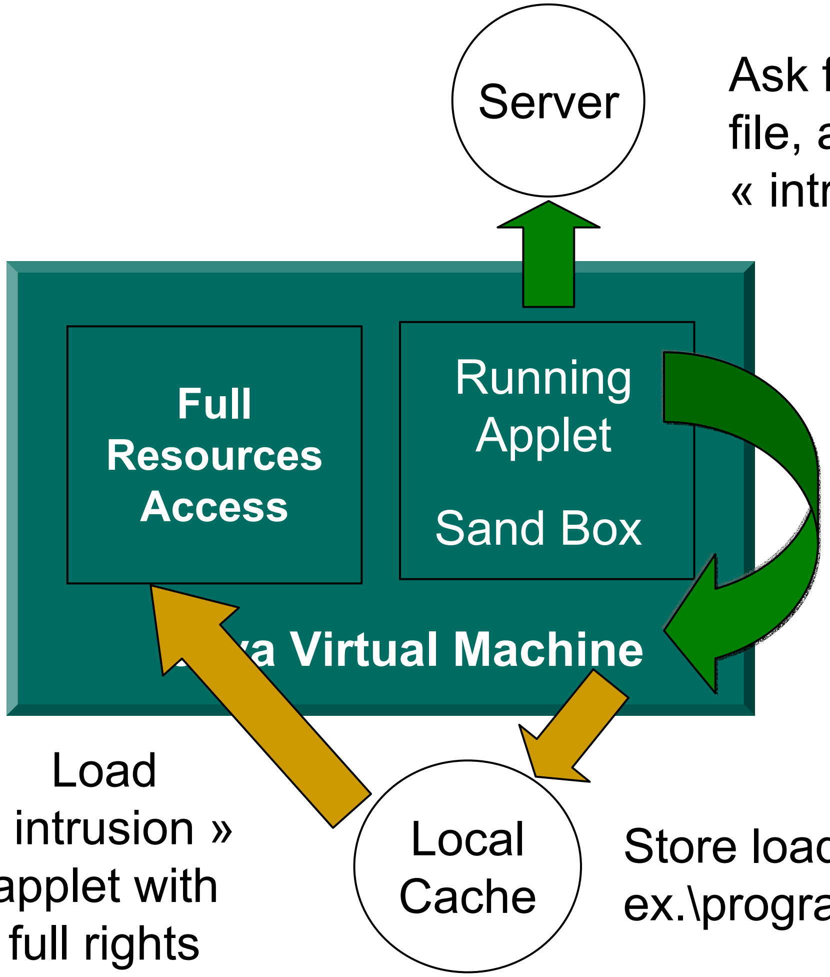
Case 2 – Slash and Burn (2)



Ask for a fake image
file, actually an
« intrusion » applet file

Store loaded file on cache:
ex.\programs\browser\cache\file47

Case 2 – Slash and Burn (2)



Ask for a fake image
file, actually an
« intrusion » applet file

Ask for class
\programs.browser.
cache.file47

JVM will try to load
\programs\browser
\cache\file47

Store loaded file on cache :
ex.\programs\browser\cache\file47

Load
« intrusion »
applet with
full rights

Fix ? No « / » or « \ » as the first letter of a class name

Nobody's perfect

- Other security flaws were discovered (e.g. wrong casting, signature spoofing, Class Loader creation by applets etc.)
- Many of these security flaws permitted break in and FULL control of the JVM (!)

Even a very good security model can be break down : by implementation bugs !!!

V. Future directions in Java Security Model

Future directions in Java Security Model

- Arbitrary Grouping of Permissions
  - e.g. group FilePermission + SocketPermission
- Subdividing Protection Domains
  - e.g. divide the « big » system domain in sub-domains with particular rights
- Running applets with signed content
  - e.g. signed images, pictures

Conclusion

- Java 2 provides a highly customizable security model for a large scale of security purpose use
- A continuous evolution towards a higher granularity of the security model and a better security management

For future reading…

Sun Microsystems Java Security Spec:

- http://java.sun.com/j2se/1.4.1/docs/guide/security/index.html

Online Documents:

- http://www.securingjava.com
- http://www.cs.princeton.edu/sip/pub/index.php3

Literature:

- Li Gong, « Java Security »